Model-Based Virtual Commissioning of a Manufacturing Cell Control System

A Thesis Submitted to the Department of Production Engineering in Partial Fulfillment of the Requirements for the Degree of Master of Science in Production Engineering and Management at the School of Industrial Engineering and Management Royal Institute of Technology (KTH)

September, 2013

By

Filmon Y. Andemeskel

Production Engineering, KTH
Abstract
One of the preferred options of manufacturing organizations to survive in today’s market is to automate, increase flexibility and configurability of manufacturing lines. Control systems and Programmable Logic Controller (PLC) programming are quintessentially important part of an automation system. However, PLC programming and debugging takes time and is error prone; consequently, there has being a growing need for quick development of PLC programs and inexpensive code verification and validation methods. To meet the needs, this thesis paper presents a method of virtual development of control system for a fully automated manufacturing cell. The cell has two robots and five other machines, which each machine operating modes are modeled in MATLAB environment and PLC code is generated from the developed models. The model and their associated PLC code are verified and validated in virtual environments. Step-by-step development, verification and validation approaches are presented and argued. Results show that few hours of modeling efforts can generate thousands of lines of code; hence this method is expected to significantly reduce time, development efforts and costs associated with verification and validation of PLC code.

Keywords: PLC Code Generation, Simulink/Stateflow Model, Verification and Validation, Manufacturing Cell.
Abstrakt


Nyckelord: PLC Code Generation, Simulink/ Stateflow Modell, Verifiering och validering, tillverkning Cell
Acknowledgement

Someone who supports you to taste the pleasures of the thing you ever dreamed of doing deserves beyond acknowledgement. Truly, my deep humble respect is to those who nursed, clothed, sheltered, cooked, cleaned, washed my stuff, and showed me around while at KTH. I might be the oldest student of the class I later joined; but better said: I was spoiled like a sensitive baby at home. The fact that I still don’t know how to do shopping and where to find a spoon in the cabinet tells a lot. During my stay in this strange land, they made home away from home. I never came across a synonym to “thank you”; so I say thank you! And they know who I am talking about.

Prof. Daniel Semere, my thesis advisor, disembarked me out of a stagnant ship, which I loved more than any other ship. Despite my refractoriness to relinquish the ship, he helped me sail into a new and dynamic one. I thank him most for that. Our extended discussion about the old ship were intellectually rewarding moments for me.

Jerzy Mikler, Ove Bayard and Farazee Asif I thank them for their email replies. Their replies helped me a lot—I mean seriously a lot, and they know how! Further, my programme was not just like any other programme of the department; so I deeply appreciate the department instructors and staff for their efforts to initiate and run the programme, and do everything in between.
List of Figures

Figure 1. Thesis Methodology ........................................................................................................ 4
Figure 2. The layout of the considered Manufacturing cell ..................................................... 5
Figure 3. Centralized Form, Proper Hierarchical Form, Modified Hierarchical Form, and Heterarchical Form ............................................................................................................. 10
Figure 4. Stateflow model representation of a motor’s on and off modes ......................... 15
Figure 5. Stateflow model representation of two parallel machines with two motors each ...... 15
Figure 6. Truth table logic for machine B ................................................................. 16
Figure 7. Stateflow model decision pattern of Machine B ................................................. 17
Figure 8. Stateflow model representation of a two motor system ..................................... 18
Figure 9. Stateflow model representation of two motors using Bottom-up-modular approach ... 18
Figure 10. Top-down model development approach for a two-motor system .................. 19
Figure 11. IRC5_STARTMAIN, IRC5_START and IRC5_MOTORSON FB blocks of IRC5 PLC .......... 21
Figure 12. Stateflow connected other Simulink blocks for simulation and validation purposes ... 23
Figure 13. Declaration part of a motor on/off PLC program in CoDeSys environment .......... 25
Figure 14. Routine part of a motor on/off PLC program in CoDeSys environment .............. 26
Figure 15. Input and output signal visualization in CoDeSys environment ......................... 29
Figure 16. Stateflow model representation of the motor states with a timer ....................... 33
Figure 17. TON function block diagram (Turn-on delay), timer set to 5 seconds ............... 34
Figure 18. Stateflow model representation of conveyor modes ........................................ 35
Figure 19. Stateflow model representation of conveyor modes ........................................ 37
Figure 20. Stateflow model representation of lathe machine Routine mode .................... 37
Figure 21. Stateflow model representation of major Robot A modes ................................ 39
Figure 22. Stateflow model representation of Robot A startup mode ................................ 40
Figure 23. Stateflow model representation of routine Robot A modes ............................. 40
Figure 24. Stateflow model representation of Robot A stop, wait and grip/release modes .......... 41
Figure 25. Stateflow model representation of Robot A motion to pallet and Conveyor A ........ 41
Figure 26. Stateflow model representation of major modes of Robot B ........................... 42
Figure 27. Robot B first cycle with lathe cycle time shorter than that of milling machines ...... 43
Figure 28. Stateflow model representation of Robot B routine operations mode .................. 44
Figure 29. Stateflow model representation of Robot A stop, wait and grip/release modes .......... 44
Figure 30. Stateflow model representation of motion of Robot B to conveyors A and B ........ 45
Figure 31. Stateflow model representation of Robot B stop, grip, and release modes along decision logic ................................................................................................................................. 46
Figure 32. Stateflow model representation of motion of Robot B to Lathe ......................... 47
Figure 33. Stateflow model representation of motion of Robot B to Milling A ..................... 48
Figure 34. Stateflow model representation of motion of Robot B to Milling B ..................... 48
Figure 35. Stateflow model representation of Robot B down mode .................................. 49
Figure 36. Stateflow model representation of Robot B shutdown mode ............................ 50
Figure 37. Stateflow charts and other Simulink blocks representing machines of a manufacturing cell and a supervisory controller ................................................................. 51
Figure 38. Stateflow truth table representation of manufacturing system wide alarm .......... 51
Figure 39. Truth table logic for system wide alarm .......................................................... 52
Figure 40. Top level Stateflow representation of manufacturing cell modes .................. 53
Figure 41. shutTable: Truth table logic for system shut down ....................................... 54
Figure 42. Manufacturing cell System Routine model ..................................................... 54
Figure 43. Stateflow model representation of Robot A routine and shut down modes .......... 55
Figure 44. Stateflow model representation of manufacturing cell down mode ............... 55
Figure 45. Sample test case for Milling B Model ............................................................. 59
Figure 46. Part of test bench input values of lathe code in CoDeSys Environment ............ 60
Figure 47. Simulink Manual Switch and other blocks connected to a Stateflow chart .......... 61
Figure 48. Sequential flow chart for lathe code validation: during Routine Step .............. 62
Figure 49. Lathe function instance invocation input parameters during steps Init and Routine... 63
Figure 50. Functionality test of the conveyor code ........................................................... 63
Figure 51. Functionality test of the Lathe code ................................................................. 64
Figure 52. Functionality test case of the Robot A code ..................................................... 64
Figure 53. Functionality test case of the Robot B .............................................................. 65
Figure 54. Manufacturing cell monolithic code functionality test: output ....................... 66
Figure 55. Manufacturing cell monolithic code functionality test: input ......................... 67

List of Tables
Table 1. Design Structure Matrix for signal flow without and with holistic considerations .... 6
Table 2. Number of input and output signals, and states per model .............................. 34
Table 3. Simulink Design Verifier Tests Report ............................................................... 58
Table 4. Statistics of generated PLC codes from 5 different models ............................. 59
Table 5. Model/Code and Rule relationship table ............................................................ 61
Table of Contents

Abstract..................................................................................................................................... i
Abstrakt..................................................................................................................................... ii
Acknowledgement .................................................................................................................. iii
List of Figures ............................................................................................................................ iv
List of Tables ............................................................................................................................... v
Chapter One ..................................................................................................................................... 1

Introduction ..................................................................................................................................... 1
1.1 Research Questions ..................................................................................................................... 2
1.2 Need for Research ...................................................................................................................... 2
1.3 Purpose ........................................................................................................................................ 3
1.4 Delimitations ............................................................................................................................... 3
1.5 Methodology .............................................................................................................................. 4
1.6 Manufacturing Cell Layout and Process Description ............................................................. 5
1.6.1 Manufacturing Cell Layout ................................................................................................... 5
1.6.2 Manufacturing Cell Process Description ............................................................................. 5
1.7 Notation of States and Variables ............................................................................................... 7
1.8 Thesis structure .......................................................................................................................... 7

Chapter Two .................................................................................................................................... 9

Automation and Virtual Commissioning: A Background ........................................................... 9
2.1 Automated Manufacturing Systems ............................................................................................ 9
2.1.1 Manufacturing Control System Architecture ...................................................................... 9
2.1.2 Programmable Logic Controller .......................................................................................... 10
2.1.3 Flexible Manufacturing System .......................................................................................... 11
2.2 Finite State Machines ............................................................................................................... 13
2.3 Modeling in MATLAB Environment ........................................................................................ 14
2.3.1 Stateflow Chart .................................................................................................................... 14
2.3.2 Stateflow Development ....................................................................................................... 14
2.3.3 Graphical Representation ..................................................................................................... 15
2.3.4 Decision Logic ..................................................................................................................... 16
2.3.5 Modeling Approaches .......................................................................................................... 17
2.3.6 Simulink PLC Coder and Code Generation ........................................................................ 20
2.4 Function Block Diagrams .......................................................................................................... 21
2.5 Model Verification and Validation ........................................................................................... 21
2.5.1 Model Verification ............................................................................................................... 21
2.5.2 Model Validation .................................................................................................................. 22
2.6 Automatic PLC Code Generation Considerations .................................................................... 23
2.7 CoDeSys as Verification and Validation Tool .......................................................................... 24
2.8 Code Verification and Validation .............................................................................................. 24
2.8.1 Interpreting a Generated Code ............................................................................................ 24
2.8.2 Code Verification .................................................................................................................. 26
2.8.3 Code Validation ................................................................................................................... 27
2.9 Summary .................................................................................................................................... 29
Chapter One

Introduction

There are several factors that manufacturing organizations should consider and master to survive in today’s market. The critical factors for survival of the organizations are short time to market, product varieties, customized products, low and fluctuation in volumes, and low price (Bi, Lang, Shen, & Wang, 2008). One of the preferred options of the organizations to meet the needs is to automate, increase flexibility and configurability of manufacturing lines. Specifically, automation systems are one the most preferred tools of production of manufacturing organizations and is a centibillion market (IMS Research, 2012). The most widely used control systems for automated shop floor processes is Programmable Logic Controller (PLC); with such need in place, organizations might have to program and reprogram their automated manufacturing lines several times. Furthermore, the developed codes need to be verified and validated in real systems. However, PLC programming, implementation and verification (most often done manually) takes time and is cost inefficient (Ko, Park, & Wang, 2008; Flordal, Fabian & Akesson, 2004; Hoffmann & Maksoud, 2010; Pan, Polden, Larkin, Van Duin, & Norrish, 2012; Sharma, 2013).

To improve development time and reduce associated costs, the paper presents a Model-based virtual commissioning of control systems. In frame work of this thesis, Model-based virtual Commissioning refers to the process of virtually capturing and representing all relevant modes of a system in block forms in such a way the generated PLC code can control a real system. The model and code correctness and applicability is checked in virtual environment. In conventional approach, integration of manufacturing cell components cannot be done before the components are built, possibly embedding design problems and faults before first system start up (Hoffmann & Maksoud, 2010). Using virtual commissioning, however, the development time and cost can significantly be reduced (Hossain & Semere, 2013). This approach when specifically used for programming robots, usually referred as Offline Programming, have proven to be more efficient and cost-effective for coding complex systems and production of large volumes (Pan et al., 2012).
This thesis paper, presents models developed in MATLAB’s model development environment—Simulink/Stateflow, for use in control of a manufacturing cell. The manufacturing cell has two robots and three CNC machines and two conveyors; Stateflow models were developed for each machine. Stateflow chart, as high level modeling language, is relatively easy to understand when compared with the lower level languages, like PLC language. In line with the argument by Alexandru (2012), the models presented in this paper are “as complex as necessary, and as simple as possible.”

1.1 Research Questions

a. Can a valid PLC code generated from a model in MATLAB environment be used to virtually commission a manufacturing cell control system?

b. How can models be developed and the generated code validated for commissioning of a manufacturing cell?

1.2 Need for Research

Several points can be stated in relation to the needs for research in virtual commissioning of a manufacturing cell control system. To mention few:

a. Virtual verification and validation of MATLAB models and PLC code are not well studied (Mazzolini, Brusaferri & Carpanzano, 2011).

b. Commissioning of industrial robots by a generated PLC code from MATLAB is not well studied.

c. Conventional integration of industrial robots costs 3-10 times of the capital cost of the robots (Brooks, 2012). The programming of industrial robotic system for a specific application is very difficult, time-consuming, and expensive, a major hurdle for automation using industrial robots (Pan et al., 2012).

d. PLC code manual programming and debugging takes time and is error prone, (ko et al., 2008; Flordal et al., 2004; Hoffmann & Maksoud, 2010; Sharma, 2013).

e. Most robot related accidents occur during robot programming (OSHA, 1999).

f. Need of flexible manufacturing systems to react to variation in demand and product variety quickly.
1.3 Purpose

The purpose of this thesis is to generate a valid PLC code from models on MATLAB environment for virtual commissioning of a manufacturing cell control system. The main purposes of the paper can be summarized as follows.

a. Explore methods for automatic PLC code generation from models on MATLAB environment.

b. Illustrate the benefits of virtual modeling, code generation, and verification and validation techniques.

c. Develop models that represent conveyors, lathe machines, milling machines, and robots on MATLAB environment.

d. Set a way to integrate the developed models in such a way to represent a manufacturing cell.

e. Verify and validate the developed models in MATLAB environment.

f. Generate PLC code from the manufacturing cell models.

g. Verify and validate the generated code in CoDeSys environment.

1.4 Delimitations

The following points were used as delimitations:

a. The research considered a hypothetical manufacturing cell; no real cell was associated with model. Only virtual development of models, verification and validation of codes was considered.

b. The exact coordinate of the machines in a shop floor environment and orientations of a product were not considered. Moreover, no specific method of setting the coordinates for data communication with robot specific program was considered.

c. The models developed for the lathe and milling machine tools are only meant to capture the modes of the machines that directly affect the manufacturing cell.

d. Since the version of MATLAB used for modeling does not support absolute time for an IDE CoDeSys, the use of timers was largely neglected. However, ways to circumvent the limitation is proposed.

e. Scheduling and capacity planning of the manufacturing cell were not considered.
f. Since no real controllers were used, emulation and real time testing were not done; instead the results are validated by 2D visualization in CoDeSys and Stateflow virtual environments.

g. Manual mode of the machines was not considered.

1.5 Methodology

There were four major steps followed in this paper: The first part included rules and requirements development. The rules were developed in such a way to capture the common operational behaviors of typical manufacturing systems.

The second part included manufacturing cell model development, verification and validation in Simulink/Stateflow. Two methods were used to develop models for the manufacturing cell. The first models were developed using top-down model development approach and the other using bottom-up approach. Using the first approach, models that can represent the seven machines were developed individually and later integrated accordingly. And using the second approach, the major manufacturing cell modes were considered and then decomposed by adding more details of the models of the cell.

Part three included preparation of input and output signal levels, followed by model validation by referring the signal levels. The decisions on related to model validation were done by checking if the rules and requirements developed in part one were met or not. A similar procedure was followed for code validation.

![Thesis Methodology](image)

Figure 1. Thesis Methodology
The last part included code generation, verification and validation in CoDeSys environment. Once models were developed, verified and validated, PLC code was generated from the Stateflow model and verified using Simulink PLC Coder in CoDeSys. The generated code was validated using 2D visualization technique in CoDeSys environment.

1.6 Manufacturing Cell Layout and Process Description

1.6.1 Manufacturing Cell Layout
The manufacturing cell has two industrial robots, two conveyors, two milling machine and a lathe machine. The first robot (Robot A) transports a work piece from a pallet to the first conveyor (Conveyor A), and the second robot picks the work piece from Conveyor A to lathe, from lathe machine to first milling machine (Milling A) or second milling machine (Milling B), finally drops the machined part to second conveyor (Conveyor B).

![Diagram of the manufacturing cell layout](image)

Figure 2. The layout of the considered Manufacturing cell

1.6.2 Manufacturing Cell Process Description
The manufacturing cell considered in this paper is a fully automated cell; hence during routine operations, no human intervention is expected. The two robots are used to transport a product to and from the machines. Robot A drops at conveyor A upstream whenever there is no product; however, the conveyor should stop when the product reaches the end of the conveyor downstream so that the Robot B would pick the product from a specific point at all times. Similarly, at Conveyor B, Robot B drops a finished product as far as there is no product on the
specific dropping point. The three machine tools’ door open and close by their internal control system; when the doors are open, Robot B picks and drops at the machine tool chuck. If system shut down is signaled, the manufacturing cell’s machines shutdown in orderly fashion.

1.6.2.1 Main assumptions

The main assumptions for modeling the cell are: All machines are endowed with asset monitoring support sensors. The conveyors have two infrared sensors at both upstream and downstream ends, one each; the machines tools have internal sensors that indicate whether machining is done, door obstacle is present, product is loaded on chucks and door limit switches; and the robots have an internal inverse kinematics and optimal point to point calculation methods that PLC program is used only to provide the appropriate coordinates. Conveyor B is expected to move the dropped from Robot B product to downstream immediately. When the robot returns with another product the target position at the conveyor will not have been occupied. If the conveyor has not moved the robot, meaning that the conveyor has stopped, then an alarm is set, stopping the whole manufacturing cell.

1.6.2.2 Data Exchange among Machines

The manufacturing cell process presented in this paper is highly coupled, information wise. The coupling is better is explained using Design Structure Matrix shown in Table 1. Since two transporters are involved to pick and drop to/from five machines and a pallet, the information required by almost all machines is iterative. “The complexity of a robotic cell increases with the number of robots and number of alternative paths” (Flordal et al., 2004); in this paper, to reduce complexity, Robot A is limited to three major paths and Robot B to eleven major paths. The data exchange among machines is assumed to be carried out by a PLCbus.

Table 1. Design Structure Matrix for signal flow without and with holistic considerations
1.7 Notation of States and Variables

In this paper, two approaches of model development were used. In first model around 250 and in the second model around 300 state and variable names were used; one or more output signal(s) of a model are inputs to one or more model(s). Hence, state and variable notation consistency is important.

For simplicity and uniformity, UpperCamelCase was used for naming the state names and lowerCamelCase for identifying the variables. However, whenever there was a need to use the name of the robots first, UpperCamelCase was used. Some outputs that were meant to indicate the status of the states was notated in uppercase letters.

For example, signal variable name RAOnTarget refers to the input signal that Robot A is on Target; the state RAMovingToConvA represents the motion of Robot A to Conveyor A and the output signal of this state is represented by an acronym RAMTCA. One of the two models used in this paper was developed with intention of downloading the generated code to a single controller, hence each and every variable of two conveyors, two robots, and two milling machines were identified with a unique name.

Moreover, Simulink® PLC Coder™ Simulink®, Stateflow®, MATLAB®, and Simulink Verification and Validation™ are trademarked products of The MathWorks, Inc. Hereafter, however, will be referred as Simulink, PLC Coder, Simulink, Stateflow, MATLAB, and Simulink Verification and Validation, respectively. Similarly, the terms “code” and “PLC code” refer to the automatically generated PLC code from the Stateflow charts using Simulink PLC Coder. Further, the terms “mode” and “state” are frequently used in the paper; mode refers to the mode of a machine, and the corresponding mode representation in Stateflow environment is referred as state.

1.8 Thesis structure

The paper is organized in five chapters:

- Chapter one introduced the principles of model-based virtual Commissioning of control systems, the needs for, purpose, delimitation and methodology of this research paper. This chapter also introduced the considered manufacturing cell layout and processes involved.
• Chapter two reviews and discusses literature in support of the approach used for model and code development, verification and validation.

• Chapter three introduces to the rules used for modeling and discusses step-by-step development of the models.

• Chapter four presents and discusses verification and validation of the results of the modeling approaches and the associated code.

• Chapter five concludes and forwards recommendations for future work.
Chapter Two

Automation and Virtual Commissioning: A Background

This chapter reviews and discusses literature in support of the tools and approaches used for modeling and code generation in this thesis paper. The chapter is organized in the following order: section 2.1 discusses automated manufacturing system in relation to PLC controllers. Sections 2.2 to 2.5 outlines the model development steps for purpose of PLC code generation, and model verification and validation. Sections 2.6, introduces how to interpret and map a PLC code by referring the models. Section 2.7 discusses considerations that must be taken for automated code generation. And section 2.8 argues methods taken by this thesis paper for verification and validation of the models and generated codes with more details.

2.1 Automated Manufacturing Systems

Automated manufacturing systems are one the most preferred tools of production of manufacturing organizations and key indicator of a country’s GDP; research shows that the automation systems market is likely to exceed USD 200 billion by 2015 (IMS Research, 2012). The goal of automation is to optimize productivity in the production of goods and delivery of systems by using (manufacturing) control systems and information technologies; the tools of automation vary from distributed control systems, Human Machine Interface (HMI) to Numeric control system (“Automation”, 2013). These tools are further discussed below.

2.1.1 Manufacturing Control System Architecture

“The control system of an automated manufacturing system coordinates and directs the parts handling and processing activities that transform raw materials into finished products.” (Diltis, Boyd, Whorms, 1991, p. 80). The control system effectiveness is dependent on control architecture; hence, proper choice of control architecture is important. Moreover, control architecture can exist in 4 basic forms (Diltis et al., 1991):

1. **Centralized Form**: one control unit controls several machines’ controllers; this architecture can do several tasks ranging from job tool availability checking to sequencing operations.

2. **Proper Hierarchical Form**: characterized by master (supervisor) and slave (subordinate) controller architecture. The architecture has mainframe computer (at the highest level),
minicomputer (at intermediate level), and microcomputers (at the lowest level). The information becomes more detailed from top down.

3. **Modified Hierarchical Form**: the strict master-slave relationship in Proper Hierarchical Form is reduced. There is peer to peer coordination between subordinates, which avoids continues supervision by the master controller and introduces some level of autonomy.

4. **Heterarchical Form**: this architecture does not use master/slave relationship, but full local autonomy at the level of peer to peer. This is possible by connecting each controller to a LAN.

![Diagram](image)

Figure 3. (A) Centralized Form, (B) Proper Hierarchical Form, (C) Modified Hierarchical Form, and (D) Heterarchical Form (Diltis et al., 1991)

Moreover, a new architecture called Agent Based Control can perform intelligent tasks and control units interact in a collaboratively; this can be lead to artificial intelligence like decision making capability (Leitão, 2009). But an intermediate architecture between the two extremes of control architecture (hierarchical and intelligent autonomous systems) is preferred for practical Flexible Manufacturing System / Reconfigurable Manufacturing Systems (Bi et al., 2008). In this paper, two models were developed with central form and Modified Hierarchical Forms in mind.

### 2.1.2 Programmable Logic Controller

Today’s PLC have evolved to handle motion control, process control, distributed control systems and networking, making their performance approximately equivalent to desktop computers. However, unlike other general purpose computers, PLC is designed immune to fluctuation in electrical noise, resistance to vibration, impact, dust, humidity, and temperature. Furthermore, modern PLC can be programmed in different languages like the IEC 61131-3 languages: *Function Block Diagrams (FBD)*, *Ladder Diagram (LD)*, *Structured Text (ST)*, *Instruction List (IL)* and *Sequential Function Chart (SFC)*. Different software’s can be used to program and debug PLC
code. Once the correctness is verified, the code can be downloaded from a computer to a controller through Ethernet, RS-232, RS-485, RS-422, EIA-485, Modbus, or BACnet. (“Programmable Logic Controller”, 2013)

Normally, PLC programs work in closed loop. To reduce scan time, the programs are sometimes segregated to programs for setting up and programs for normal routine. In addition, whenever there is a need to increase input/output lines, additional modules can be inserted into PLC racks (or slots) easily, reducing wiring costs. Moreover, a PLC may need to interact with humans; Using Human Machine Interface (HMI), a PLC can interact with humans easily. A typical HMI includes lights, texts, and buttons. Push buttons, limit switches, and photoelectric sensors can provide discrete signals that can behave as binary switches, either True(1) or False (0). A 24 V DC PLC, for example, may use values above 22V DC to represent True and 2V DC to represent False, while intermediate values are undefined. (“Programmable Logic Controller”, 2013)

2.1.3 Flexible Manufacturing System

Flexible Manufacturing System (FMS) is a manufacturing system that responds to changes in (predicted or unpredicted) demand at least by its ability to produce new product and change order of operations executed on a part, and/or multiple machine to do the same operation on a part; an FMS should to able absorb volume, capacity or capability changes. The advantages FMS range from reduction manufacturing time and inventory, improved quality and adaptability to CAD/CAM operations (“Flexible Manufacturing Systems”, 2013). Out of possible elements of FMS, conveyor, lathe and milling machine tools and robots were the interests of this paper.

2.1.3.1 Conveyors

A conveyor (system) is piece of mechanical handling equipment that moves materials form one location to another. These systems are available in varieties and are widespread across range of industries (“Conveyor System”, 2013), out of which Flexible conveyor system is the interest of this paper. Further, some control related recommendation by OSHA (2001) worth stating are: when conveyors are in series, if one machine conveyor stops, all should stop; and one should equip conveyors with emergency stop, which would require manual resetting before resuming.

2.1.3.2 Machine Tools

A typical FMS is more likely to use Machine tools. Machine tools are machines for shaping rigid materials by cutting, boring, grinding, sharing or other forms of deformation. Most machine
tools are controlled manually and automatically. Modern machine tools use a Numerical Control endowed with processors; these machines are commonly known as Computerized Numerical Control (CNC) machines. CNC machines can machine starting from the simple to the most complex, by strictly following sequence of codes (“Machine Tool”, 2013).

2.1.3.3 Industrial Robots
Most of the industrial robots are programmed with specific dedicated languages. However, the new generation of ABB industrial robots can be controlled by PLC program. Since most of the manufacturing equipment in today’s manufacturing organizations is controlled by PLC, spares PLC programmers from training a new language (ABB, 2010). Moreover, most of the robot related accidents occur during robot programming and maintenance, not during routine robot operations (OSHA, 1999); hence there is a need for offline programming and validation.

However, robot modes can be difficult to represent with traditional programming approaches, condition and time based decision logic can be very complex, and debugging the design can be challenging (Sharma, 2013). The complexity can be reduced by strictly following predetermined sequence of steps (Flordal et al., 2004). Some researchers like Park, Park, & Wang (2009), Falkman, Helander, & Andersson (2011), Flordal et al. (2004), Moura and Guedes (2012), and Hossain & Semere (2013), have showed the benefits of programming robots by PLC and validation in virtual environment (using different Software’s).

2.1.3.3.1 Programming and Coordinate system
Programming of industrial robotic system for a specific application is very difficult, time-consuming, and expensive, a major hurdle for automation using industrial robots. Industrial robots can be programmed either by Online or Offline Programming. During online the robot is led through or walk through, which the end effector coordinate and postures are recorded while an operator manual manipulates the robot. This method need low programming skills, but there is no way of editing the recorded coordinates and postures. In Offline Programming uses different software to develop the code on PC, which is then downloaded to the controllers. This method is much more flexible than Online Programming method (Pan et al., 2012).

All ABB robots uses the Right Hand Rule; ABB defines a coordinate the base of the robot known as the base coordinate system The X-Y plane is the same as the base mounting surface, which X-axis points forward from perspective of the robot. Tool center point location in reference to this
coordinate system can be used to program robot (Lehtla, 2008). Normally, ABB robots are programmed position to position (often referred as pos-to-pos). “The path between these two positions is then automatically calculated by the robot” (ABB, n.d., p.29).

2.1.3.3.2 The Issue of Deadlock
The qualities that make an FMS flexible make it complex. Deadlock is difficult to predict and may bring the system to complete halt. Clearing a deadlock might further induce high labor cost; hence there is absolute importance to avoid deadlock (Hartonas-Garmhausen, Clarke, & Campos, 1996). However, MATLAB tools have exhaustive testing tools that can check dead logic, conflicting signals and unreachable states (The MathWorks, 2002, 2013a), which might reduce the probability of deadlock. The tools are discussed in subsection (2.5.1.1.).

2.1.3.3.3 The Issue of Collision
Industrial robots can operate so close to each other that collision may happen. To avoid collision the shared spatial volume must be supervised in such a way one robot can access this particular volume in any given time (Flordal et al. 2004). In this paper, however, the two robots do not share any spatial volumes.

2.1.3.3.4 The Issue of Scheduling
Scheduling a manufacturing cell with multiple robots is not an easy challenge. Even under very special conditions (e.g., all the processing times equal or all the machines are identical), the problem (of scheduling a robotic cell) is NP-hard (Levner, 2010). A survey by Levner (2010) presents several research efforts to find an optimum scheduling and solutions to a no-wait problem of robotic cells, yet with several limitations. In this paper, the manufacturing cell scheduling is not considered; hence some machines might have to go to corresponding waiting modes, depending on cycle time of each of the machines.

2.2 Finite State Machines
Finite State Machine is defined as “A model of computation consisting of a set of states, a start state, an input alphabet, and a transition function that maps input symbols and current states to a next state” (NIST, 2008, p. 595). The principles of Finite State Machine were developed with intention to fit human beings’ frame of mind (Harel, 1986).

Finite State Machine is a representation of event driven reactive system (The MathWorks, 2010). Reactive system is a system that reacts to internal and external stimuli (Harel, 1986).
Finite state machines are those that include condition triggered actions; in the representation of a system, as the name indicates, there are finite states or modes in a machine (or a system), when condition is satisfied the machine changes its mode.

2.3 Modeling in MATLAB Environment

2.3.1 Stateflow Chart

Finite State Machine can be represented by capturing the states of the machines and transition between states to model the behavior of a system. Since states and transitions are used, the chart built by using state and transition symbols is called State transition diagram. MATLAB’s Stateflow chart is based on state transition diagram (The MathWorks, 2002).

Stateflow is defined as: “An environment for modeling and simulating combinatorial and sequential decision logic based on state machines and flow charts” (The MathWorks, 2013d, para. 1). MATLAB utilizes the Stateflow chart for machine state animation, static and runtime checks for testing design consistency and completeness before implementation. Stateflow chart can also be used for response and event driven systems, condition based logic, scheduling activities/modes (The MathWorks, 2002).

Normally, states represent the modes of a machine, which can be active or inactive at any time of the execution. Stateflow has a representation of states as parent-child relationship, which under Stateflow terms, parent is called superstate and the child as substate (The MathWorks, 2002).

When the operating modes are mutually exclusive (called Hierarchy) represented by OR-decomposition; when operating modes co-exist (called concurrency) represented by AND-decomposition; and when a mode exists one state and returns to the state that it left off out several possible modes is represented by use of history junction (The MathWorks, 2002).

2.3.2 Stateflow Development

MATLAB proposes two step basic approach of Stateflow model development (The MathWorks, 2002):

1. Stating what the interfaces are (triggering event and inputs to and output from the system).
2. Checking if there are multiple modes and the modes run in parallel.

Using this approach, in a single motor system, the input signal is a Boolean stop and the output is a Boolean motor, see figure below. The motor lives in two mutually exclusive modes MotorOn and MotorOff: linking the two modes by a condition that signal stop should change from True(1) to False (0), or vice versa, for modes to transit from one mode to another. However, this approach of development is challenging in several aspects (see section 2.3.5 for detailed discussion). Stateflow chart uses one or more of graphical, tabular or pattern methods to represent finite state machines.

![Figure 4. Stateflow model representation of a motor’s on and off modes](image)

**2.3.3 Graphical Representation**

The figure below shows anatomy of a typical graphical representation of a Finite state machine for PLC code generation. There are several syntaxes that can be used for Stateflow development to utilize Stateflow to its full capability. However, the PLC works with Boolean logic and basic mathematics (The MathWorks, 2013c), which limits the use of the Stateflow full capabilities.

![Figure 5. Stateflow model representation of two parallel machines with two motors each](image)
At the top left corner of each block is the state name, what is enclosed between square brackets is transition condition, and what is enclosed between curly brackets is the condition action. Transition can occur only when a transition condition is satisfied. When transition condition is satisfied the transition action, if available, set to a given value. The states can be mutually exclusive or parallel (TheMathWorks, 2002). For instance in the above Figure $[MB\text{StopA}==\text{true}]$ is the transition condition and $\{MB\text{StopA}=\text{true}\}$ is the transition action. The model shows and represents that the two motor in Machine B exist in two mutually exclusive states but motors of Machine A work in parallel; and the two machines themselves work in parallel.

### 2.3.4 Decision Logic

#### 2.3.4.1 Tabular Representation

Truth Table is a tabular logical decision making tool. Truth table enables the user to create functions by specifying decision outcomes for conditions and their corresponding actions without having to draw flow diagrams. Truth table can be developed inside or outside the Stateflow environment (The MathWorks, 2002). The following Figure is tabular representation of a Machine B in shown in Figure 5.

![Figure 6. Truth table logic for machine B](image-url)
2.3.4.2 Pattern Representation

Stateflow can represent common logic decision (if-else(if), nested if), iterative loop (for, while, do-while) and switch with several cases using decision logic pattern. The utilization of the pattern reduces modeling efforts to some extent (The MathWorks, 2013b). The following Figure shows an if-else(if) logic for Machine B.

Figure 7. Stateflow model decision pattern of Machine B

2.3.5 Modeling Approaches

Finite State Machine principles fall short to model complex system behavior in many aspects, like in coupled robotic-cells. The proposed extension of the Finite State Machine principle is the Extended Finite State Machine (EFSM) (Alagar & Periyasamy, 2011). Since Stateflow chart enables the representation of hierarchy and parallelism (The Mathworks, 2002) and under EFSM principles complex systems can be represented as combination of parent and offspring blocks (Alagar & Periyasamy, 2011), this thesis utilizes both Stateflow charts and EFMS principles for modeling a manufacturing cell.

Using the EFSM principles, two modeling approaches can be used: Top-down and bottom-up approach. Using the top-down approach Stateflow model is developed in hierarchical manner; using the bottom-up approach, the Stateflow model is represented in modularized manner, which the transitions are connected accordingly (Alagar & Periyasamy, 2011).
The model in Figure 8 above has two parent states, and each of which have two siblings. Each motor can exist in either On mode or Off mode. The model can be developed by following Bottom-up approach and Top-down approach.

2.3.5.1 **Bottom-up Development of Finite State Machine Models**

In scenarios like in this paper, where the two conveyors and the three machine tools interact only on incidents of alarm and stop signals, modular modeling is preferred method of model development. Using this approach, all components’ representative models are developed separately. Since most of the system model details are developed independently, the complexity in modeling complex interaction between component machines is reduced. Each component can be modeled using **Top-down model development approach** (see next subsection). The interactions between components are later composed with the models, and this approach is likely to achieve completeness of the whole model (Alagar & Periyasamy, 2011).

A modular model can be associated with a grey box, which all states, transitions, and variables that interact with other components are made visible, and are meaningful. Moreover, transitions can link superstates and substates; for simplicity, however, the interaction between superstates and substate can be avoided by using sufficiently detailed conditions (Alagar & Periyasamy, 2011).
To develop a model for the two-motor system, for instance, Motor A model is developed independent of Motor B, and vice versa. When both models are developed, interacting transition condition \((\text{changeMotor})\) between Motor A and Motor B is introduced. The interaction, say, between superstate \(\text{MotorA}\) and substate \(\text{MotorB.MotorOff}\), is avoided by introducing a condition action \(\{\text{motorB=false;}\}\).

### 2.3.5.2 Top-down Development of Finite State Machine Models

Using the Top-down Model Development Approach, Stateflow is developed in hierarchical manner (from abstract model to concrete model), which incrementally adds behavior until no further decomposition is possible—until terminal state. The approach is better suited for models that include shared events, variables, and associated constraints on them; as in case of robotic-cell, where a robot has to load/unload from several machines, top-down approach is a preferred method of model development (Alagar & Periyasamy, 2011) and is common for control of a single manufacturing cell (Diltis et al., 1991). Further refinement of composite states creates inverted tree form shown in figure below.

![Top-down model development approach for a two-motor system](image)

Using this approach, what is stated at the parent state is not necessary to be stated again in the child. The child will inherit the conditions from the parent state. What is stated at the state level will not be used as condition for transition. This approach will make the model terse and easy for state interruption (The MathWorks, 2002; Alagar & Periyasamy, 2011). This approach can also deflate the concern by (Leitão, 2009) that large monolithic and centralized software are expensive to implement, maintain or reconfigure; since the code is maintained and reconfigured at a model level, not at code level (see section 4.3). Moreover, the need that engineers have to visualize all states of a production line Ko et al. (2008), can easily be satisfied by following top-down approach (see section 4.2.2.5)
2.3.6 Simulink PLC Coder and Code Generation

MATLAB has an embedded model-based PLC code generator—Simulink PLC Coder. According to The MathWorks (2013c), “Simulink PLC Coder generates hardware-independent IEC 61131 Structured text from Simulink models, Stateflow charts, and embedded MATLAB functions” (para. 1). The Coder is a tool for control and algorithm design during PLC and machine manufacturing, and system integration (The MathWorks, 2013c).

As discussed subsection 2.3.1, the Stateflow charts are developed by representing all modes of a machine. Once a Stateflow chart is developed, compatibility is checked, code is generated and verified. This is done simply by right-mouth clicking on the Stateflow chart and choosing between the options “Generate code for subsystem”, “Generate code and import for subsystem” and “Generate, import and verify for subsystem”. Some Simulink models may have to be converted to a subsystem before code generation. In this paper, despite the capability of the Coder for generating codes from Simulink models, only Stateflow models are considered for code generation.

The Simulink modeling environment should be set to fixed step size (discrete—no continuous states) before code generation (The MathWorks, 2013c); one of the reasons is that the Stateflow responds to instantaneous changes in dynamic systems (The MathWorks, 2008). Since machines operate in both continuous and instantaneous changes, Stateflow should be used to control the modes of a machine and Simulink to control behavior within the modes (The Mathworks, 2008). So there is a need for appropriate link between Stateflow and Simulink blocks. However, there are several Simulink semantics that PLC Coder doesn’t support, which ranges from absolute time temporal logic, triggered subsystem, limited support for advanced mathematical functions to merged blocks (The MathWorks, 2013c).

Since Simulink PLC Coder does not support absolute time for temporal logic for CoDeSys, use of timers is not possible. (This paper uses MATLAB 2012a, with which temporal logic is possible only for Rockwell Automation RSLogix 5000 IDE). Furthermore, Simulink PLC Coder generates code in structured text format (The MathWorks, 2013c), however, several controller functions like, ABB IRC PLC based controllers for executing of several robot actions and CoDeSys’s timer On/Off, etc, use Function Block Diagrams (ABB, 2010; 35, 2007b).
2.4 Function Block Diagrams

There are several industrial robots in the market; this thesis, however, only ABB robots are considered. Recently, ABB has developed an integrated controller option, which a PLC (AC500 ABB) is mounted at the IRC5 robot controller, in one cabinet. The basic setting of the controllers pre-programmed with setting programs on the controllers; the setting programs must be uploaded from the PLC to a PC to gain access to IRC5 controller. There is an option when using the IRC5 controller to set the RAPID program as a slave and the PLC as a master (ABB, 2010), which the author of this thesis paper has kept in mind while developing the models.

Several robot actions can be instantiated using FBDs; as stated in the above section, Simulink PLC Coder generates in code in structured format, while ABB IRC5 uses FBDs, complicating the automatic code generation process. The PLC as a master setting uses FBDs such as IRC5_STARTMAIN, IRC5_START and IRC5_MOTORSON for starting the robot program and turning the motors on, respectively (ABB, 2010); but these FBDs cannot completely be modeled in Stateflow and the generated PLC need retouching (see section 3.2 and subsection 4.3.2.3.2 for details).

![Image of IRC5_STARTMAIN, IRC5_START and IRC5_MOTORSON FB blocks of IRC5 PLC (ABB, 2010)](image)

2.5 Model Verification and Validation

2.5.1 Model Verification

Once the models are developed, Simulink blocks can be connected to Stateflow chart to simulate the behavior of the model. One important step in verifying and validating a model is Simulation; several Simulink tools check syntactic and semantic errors before and during simulation. By simulating a model, the correctness of the model for simulation purpose (hence verification) and study of the model behavior (hence validation) can be done (Sargent, 1996). However it’s important to note that “Simulation can in general only reveal the presence of
errors, not guarantee the absence.” (Floral et al., 2004, p. 2). The issue of verification and validation is further discussed in section 2.8.

Since there is no structured verification methodology within the industrial automation domain, the automation engineer decides what to verify/validate (Mazzolini et al., 2011). Moreover, verification and validation of models is based on abstract ideas, which the outcome of verification and validation is neither perfect nor imperfect (Balci, 2010). Hence, this paper relies on MATLAB model verification tools (Model Advisor and Design Verifier).

2.5.1.1 Utilizing Simulink Model Advisor and Stateflow debugger
The Simulink Model Advisor can be consulted for verification and validation of the developed models. According to The MathWorks, (2013a), “The Model Advisor checks a model or subsystem for conditions and configuration settings that can result in inaccurate or inefficient simulation of the system that the model represents” (para. 1). The advisor also checks inefficient code and/or unsuitable for safety critical applications when using automatic code generation. (These checks are done as per several standards guidelines). Running the Model Advisor yields an HTML report (The MathWorks, 2013a). In addition, the Stateflow Parser performs syntax checks before simulation and Stateflow debugger dynamic checks during Simulation, thus consistently checking the correctness of models (The MathWorks, 2002). In the framework of this paper, a model is said it to be “developed”, only if the model can be simulated without any error report.

2.5.1.2 Utilizing Simulink Design Verifier
Design verifier has the following options for model verification “System compatibility”, “Detect design error in subsystem”, “Generate tests for subsystem” and “prove properties of subsystem”. Design verifier generates test cases that satisfy objectives including decision and condition coverage (the range of possible input variables set by the user). When in test generation mode, produces tests cases that satisfy specified criteria (normally by the user). Once the test is generated, returns with analysis of the result for review (The MathWorks, 2010, Mazzolini et al., 2011).

2.5.2 Model Validation
The virtual study of the behavior of a manufacturing cell can be explained by Virtual commissioning concept. The concept is when applied to industrial application means virtual
implementation and validation using virtual tools prior to the implementation of the real systems (Hossain & Semere, 2013). Further, Virtual Reality Modeling Language (VRML) using the MATLAB Virtual Reality Tool Box can be used to design a 3D virtual manufacturing cell. This is done by connecting the Stateflow model to VR-sink block where the translation, rotational motion is manipulated by the Stateflow model (The MathWorks, 2004).

The behavior of the developed models can be studied by connecting Simulink blocks to a Stateflow block; different signal levels can be changed by using Manual Switch block. The effect of the changes can be studied state in Stateflow flow diagram simulation and signal level in Scope block, simultaneously (The MathWorks, 2002).

2.6 Automatic PLC Code Generation Considerations

Any model can be developed and corresponding PLC code be generated. The models developed here in this paper are for a specific manufacturing cell with a given product and information flow in mind. However, during model development there are several factors that need to be considered.

According to Güttel, Weber & Fay, (2008), the following factors should be considered in addition to the sequencing of the routing activities---which arguably is small part of the code: Interlock logic, safety functions, start-up and shutdown sequences, emergency break for each machine. Further, the necessary functions that need to be considered are Alarms, Start up, Asset monitoring, Operation monitoring, Diagnoses, Shutdown, Communication, Tracking of work pieces, Safety circuits, Safety state, Time-out, Interlocks, and Restart.
2.7 CoDeSys as Verification and Validation Tool

Controlled Development System (CoDeSys) is a complete development environment for PLC. The first Program Organization Unit (POU) is named PLC_PRG; from this POU other POUs like function blocks and functions can be accessed. Further, CoDeSys can compile and simulates a PLC program and download to a controller. The program can be developed in one of the following Standard IEC-61131 programming languages which include IL, ST, SFC, FBD, and LD (3S, 2007b).

CoDeSys is also endowed with an Integrated Visualization Editor. This editor has visualization objects that can be utilized along the application code development in the same user interface. The editor can directly access variables in a controller. Moreover, the editor has different elements (different block shapes, buttons, etc), animations (like text display, color change, etc), and input possibilities (Toggle/Tab Boolean values, text input, etc). In many cases the editor can provide a means of complete control and visualization the PLC program (3S, 2007a). Since the output of the code can be studied from the editor, CoDeSys can be used for validation purpose. See subsection 2.8.3.2 for more details and Figure 15 for an example the integrated editor.

Simulink PLC Coder, uses CoDeSys as PLC target IDE on which a PLC code and test bench are generated. The generated code is mostly put as function block from which test bench parameters can be used as inputs to the function block (The MathWorks, 2013c). Test bench out can be then used for code verification and Visualization Editor for code validation.

2.8 Code Verification and Validation

2.8.1 Interpreting a Generated Code

The code is generated by Simulink PLC Coder and exported into target IDE, like CoDeSys. Normally, the generated code has declaration and body part. The generated code for the model shown in Figure 3, is presented in Figures 13 and 14. In addition to the declared variables, shown in the Figure 13, the following global variables are declared.

\[
\begin{align*}
\text{Chart\_IN\_MotorOn} & : \text{USINT} := 2, \\
\text{Chart\_IN\_MotorOff} & : \text{USINT} := 1, \\
\text{SS\_INITIALIZE} & : \text{SINT} := 0, \text{ and} \\
\text{SS\_STEP} & : \text{SINT} := 1
\end{align*}
\]
The variables `Chart_IN_MotorOn` and `Chart_IN_MotorOff` represent the states `MotorOn`, `MotorOff`. The behavior during and after system initialization, are declared by `SS_INITIALIZE` and `SS_STEP`. Each state of a machine is assigned a number.

![PLC program in CoDeSys environment](image)

Figure 13. Declaration part of a motor on/off PLC program in CoDeSys environment

In Figure 13, in line 001 the `ssMethodType` of type `SINT` is assigned either 0 or 1, which corresponds `SS_INITIALIZE` or `SS_STEP`, respectively. During the `SS_INITIALIZE` the program assumes the motors are off, which is corresponds to the default starting. If there is a need to reinitialize a state machine, then the `SS_INITIALIZE` should be instantiated.

The condition in line 0012 represents the first entry to the routine part of the program, during which the state (`Chart_IN_MotorOff`) is active and the motor is turned off. In line 0020 and lines after, represents during `MotorOff` state; if the condition is stop is not true, the motor makes a transition to state `MotorsOn` and expected to turn on the motors (line 24-26), but if the stop signal is true then the motor is expected to remain in the `MotorOff` state (line 30-34).
2.8.2 Code Verification

According to IEEE (1990, p. 81), code verification is defined as “(1) The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. (2) Formal proof of program correctness.” In the case of this paper, “the condition” is that the generated code should represent the model developed in Stateflow, and “the start of the development phase” is the beginning of the Stateflow model development process. Hence, code verification, in the frame work of this paper, means the comparison of input and output signals to/from a model and corresponding generated code.

Since PLC programming is often done manually, implementation and verification of PLC software is tedious task (Floral et al., 2010); there is need for automatic and efficient verification approach. To meet this need, Simulink PLC Coder simulates the developed model and automatically captures all the inputs and outputs. Each set of input signals is later used as a test case. The output of the test case in CoDeSys is compared with output of the model in Stateflow. If the outputs are functionally and numerically equivalent to the model for the same set of input signals, then the correctness of the generated code is verified, otherwise not verified (The
MathWorks, 2013c). However, verification is an internal process done by Simulink PLC Coder; the Coder can also verify the correctness of a code that wrongly modeled. Hence, there is always a need for code validation.

2.8.3 Code Validation
IEEE (1990, p. 80) defines validation as, “The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.” Since there is dependency of validation up on verification (Oberkampf & Trucano, 2006), the paper validates only those codes that are verified by Simulink PLC Coder.

Further, validation is satisfying the needs of the user; in this case, the needs of the process engineer—author of this paper. The preferred method validation is to be performed by disinterested third party (outside code developers and users), (“Verification and Validation”, 2013, Para. 1). In this paper, the code developer is Simulink PLC Coder; hence, a code generated from a validated model also needs further validation by a third party.

The generated code can be validated into steps: first step is providing set of desired input signals that produce specific expected output(s); and second step is generating random set of input signals that produce any output signal, which the outputs are studied by referring the input signals. However, for large systems complete testing is not possible (Balci, 2010). One complimentary option to handle the second step is combinatorial testing; a combinatorial testing using principle of Design of Experiments has been applied by several researchers for cheap sampling of the random signals, yet with inherent limitations (Kacker, Kuhn, Lei, & Lawrence, 2013). Nevertheless, the second step is out of scope of this paper and will not be discussed any further.

2.8.3.1 Rules as Functional Requirements
As the defined by IEEE (1990), the goal of validation is to check if a code satisfies a given (set of functional) requirements of a machine/ manufacturing cell. Functional requirements are those expectations of the engineer from the manufacturing cell or particular component of the cell. A functional requirement that a robot has to transport a stock from a pallet to a conveyor has underlying rule that the robot should only transport a stock from the pallet to the conveyor, not
in the inverse direction. A similar line of argument for software development has been discussed by (Gottesdiener, 1999).

Hence, the rules themselves are strict and main forms of the functional requirements (Gottesdiener, 1999). The functional requirements should be separated from constraints associated with function Suh (1990, 2001). For the above example, both the functional requirement and rule have the same content, except the phrase “not in the inverse direction”—which is considered as constraint of the function requirement.

Using the above argument, the paper uses the rules as functional requirement; since validation is satisfying the needs of the user, by selecting set of input signals in accordance to the rules, the generated code is expected to produce outputs that would satisfy the needs of the user, otherwise the code is not valid for use.

2.8.3.2  Visualization as a Tool for Validation

As stated in the above, the outputs of the tests cases must be closely studied; however, the options for studying the behavior of the system are limited. One option is to directly download on controllers and see the behavior on the machines, and the other option is visualize the behavior in virtual environment. The former option is costly, as put forward in section 1.3.

Control System Emulation is one the preferred method of industrial control code virtual validation. Emulation is a method of replicating and mimicking behavior components of an automation system in virtual environment (“Emulation for Logic Validation”, 2013). An intuitive visual validation in a 3D graphic view of manufacturing cell has been presented by (Ko et al., 2008); the developed method utilizes the output of a PLC program as events that trigger the next action of the digital model. Similar approach was used by (Moura and Guedes, 2012, Hossain & Semere, 2013).

A 2D visualization method can also be used without compromising the purpose of the visual validation approach. This approach uses color change for change of signal level (True or False). Figure 15 below shows change of color (representing the output of the generated code shown in Figure 3 and 13) by activating the start/stop buttons. When the program is run using there is no change of color, preventing accidental startup of the machine. When start button is pressed the program, shown in Figure 14, enters routine step (SS_STEP) and changes color to green,
indicating the output signal is to turn on the motor. When stop button is pressed color changes to gray, indicating the output signal is to turn off the motor.

Figure 15. Input and output signal visualization in CoDeSys environment

2.9 Summary

This chapter introduced the ideas and principles of automation of manufacturing systems in relation to virtual commissioning. The applicability of automation system to the machines used in this paper was introduced and discussed. The principles of Finite State Machines was presented and discussed in relation Stateflow models. The Extended Finite State Machine modeling principles were presented and argued. Based on these principles, how Stateflow models can be developed and why specific model development approaches should be used for specific application was argued. In relation to code generation, capabilities of CoDeSys environment for verification and validation purposes were presented and discussed. Finally, methods for model and code verification and validation were introduced, discussed and argued.
Chapter Three

Modeling a Manufacturing Cell Modes

This chapter presents the methods followed for modeling manufacturing cell modes in Simulink/Stateflow environment. There are three main parts in this chapter. The first part presents the developed manufacturing cell control related rules and requirements. Based on the rules and requirements, bottom-up model development approach in the second part and top-down model development approach in third part are presented and discussed.

3.1 Manufacturing Cell Control Related Rules and Requirements

Models and codes are developed to meet specific rules and requirements; in this thesis, the rules and requirements development was done in such a way the rules are independent to each other but collectively exhaustive, in line with principles of Suh (1990, 2001). Since models are developed only for intended use Balci (2010), the author of this thesis, developed these rules for purpose of controlling a manufacturing cell elements.

3.1.1 General Rules

Rule G1: To prevent accidental start up, during initial run of the program, all Boolean values should be set to “False”.

Rule G1.1: All machines should start from corresponding down modes.

Rule G2: If any machine is stopped by either by emergency button, stop button, internal error or power cut, manufacturing cell wide alarm is created; consequently all machines are stopped, leading to their respective down modes.

Rule G3: If the machines are stopped and reset button is not set, then all machines start from modes the machines previously stopped, otherwise the machines restart from the default mode.

Rule G4: All machines exit their respective down modes, if none of the factors that led to the modes are true.

Rule G5: All machines shutdown only when there is no product loaded.

3.1.2 Conveyor Specific Rule

Rule C1: Conveyor should stop when a product reaches at the downstream end of the conveyor.

3.1.3 Machine Tool Specific Rules

Rule T1: All machine tools machine only when safety door is closed.

Rule T2: If a product has exited from a machine tool, then that particular machine tool should remain open to accept a new product.

Rule T3: All machine tools open independent of the Robot’s input signal but door closes only when robot delivers a product.

3.1.4 Robot Specific Rules

Rule R1: During routine mode, robot should neglect execution of any input signals before completing execution of the prior input signals.

Rule R2: Robot B should make priority to the machine tool which product has just previously gripped away.

- For instance, the scenario when lathe open ready to accept a product and Milling A has just finished machining while the robot was at just at dropping a product at Milling B from Lathe. In this scenario the robot should go to Conveyor A to feed the lathe first.

Rule R3: Robot B should move to waiting position only if there is no product gripped; Robot A should go to waiting position if there is a product on conveyor A or no product on pallet.

Rule R4: If there is a product on the three machine tools then, Robot B should move to the Millings to leave a space for the product on the Lathe.

Rule R5: Robot B should move to a particular machine only if the succeeding machine, according to product flow, is ready for the robot. Robot A should move to the pallet if there is a product on pallet and to Conveyor A if there is no product at the upstream of the conveyor.
• Conveyor A is ready for Robot B, if there is a product at the downstream of the conveyor.

• The three machine tools are ready for Robot B if their respective doors are open.

• Conveyor B is ready for Robot B, if there is no product is upstream target position.

**Rule R6:** If both milling machines are ready at the same time, priority should be given to Milling A.

**Rule R7:** The robots drop a specific type of product (stock, semi-finished, or finished) at specific machine.

• For example, the robot should not put a turned product from Lathe and drop it at Conveyor B

**Rule R8:** The robots should move to home position before shutting down.

### 3.2 Models for Function Block Diagrams

Several functions of CoDeSys are based on FBD. Timers, for instance, are important part of automation, which are represented as Function blocks in CoDeSys. But since Simulink PLC Coder does not support absolute time for temporal logic (The MathWorks, 2013c), there is need to set models for manual insertion of instances invocation of the FBD. This limitation was circumvented by considering the modes of the machine associated with timer block as a state of the machine—during delay the machines enter to a new mode.

![Stateflow model representation of the motor states with a timer](image)

Figure 16. *Stateflow model representation of the motor states with a timer*

By closely studying several generated codes, one could detect the patterns of the generated codes. The only pattern used by the Coder was that the output signals were set at the end of the
program; hence, declaring an output, \textit{timerTrig} in the above mode, for instance, was declared as output at Stateflow environment. Further, declaring \textit{timerInst} as \textit{timer on (TON)} and adding two lines at the end of the program at the CoDeSys environment, the timer was set up without compromising the integrity of the code. The condition \textit{waitingDone} was set to True after some time had elapsed. Several tests were done to check the invocation of the timer; for example, for the case Figure 15-17, the results showed that motor-on signal appeared 5 seconds later. Similar approach of modeling robot modes is discussed in subsections 3.3.1.3.1 and 3.3.1.4.7.

![Figure 17. TON function block diagram (Turn-on delay), timer set to 5 seconds](image)

\begin{verbatim}
timerInst: TON; (*timerInst Declaration *)
timerInst(in:=timerTrig, pt:=t#5s); (* timer trigger*)
waitingDone:=timerInst.Q; (*sets output of timer*)
\end{verbatim}

### 3.3 Manufacturing Cell Model Development

This section presents two model development approaches: \textit{Bottom-up-modular} and Top-down approach. The choice of the approaches is based on the argument presented in subsection 2.3.5. The following table shows the number of input, local and output signals, and states used during model development.

<table>
<thead>
<tr>
<th>Model/Number of</th>
<th>Input signals</th>
<th>Local signals</th>
<th>Output signals</th>
<th>Superstates and Substates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conveyor Model</td>
<td>9</td>
<td>0</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Machine Tool Model</td>
<td>12</td>
<td>1</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>Robot A Model</td>
<td>21</td>
<td>10</td>
<td>17</td>
<td>26</td>
</tr>
<tr>
<td>Robot B Model</td>
<td>27</td>
<td>11</td>
<td>23</td>
<td>49</td>
</tr>
<tr>
<td>System Alarm Truth Table</td>
<td>7</td>
<td>8</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Monolithic Model</td>
<td>77</td>
<td>50</td>
<td>75</td>
<td>103</td>
</tr>
</tbody>
</table>

#### 3.3.1 Bottom-Up-Modular Model Development Approach

The procedures for model development were discussed in relation to EFSM. This paper has adopted a name “Bottom-up-modular” model development approach in accordance to the
principles of EFSM. This approach develops models starting from manufacturing cell machines separately. Once the machine models were developed, truth table was used to indicate the status of the manufacturing cell.

3.3.1.1 Conveyor Models

The model presented in this subsection, model for Conveyor A, can also work for Conveyor B. Conveyor A is considered to exist in either of three mutually exclusive modes: Down, Routine, and Shutdown modes. Upon start up the conveyor should enter the safety stop mode; this requirement was represented by state ConvADown and signal output convADwn at HMI for asset monitoring. During this mode the motor should be turned off (set by convAMotor=false).

Only when an operator pushes start button, and emergency, stop and manufacturing cell alarm signals are not sensed, the conveyor should make a transition to Routine mode, which was represented by ConvARoutine state. But the conveyor can return back to the down mode if either emergency, stop, power cut, or alarm signals are sensed (thus, these conditions were represented by convAEmStp, convAStop, convAPowerRelay, or sysAlarm).

![Figure 18. Stateflow model representation of conveyor modes](image)

During routine mode, the conveyor belt rolls or halts according to the presence of a product at the downstream end of the conveyor; if the product is detected, say by infrared sensor (represented by prodOnCADwnStrm), rolling stops and waits for Robot B for pick up (represented by convAHalt), in the case of Conveyor A. Once the product is picked up the infrared sensor should be set to False and the conveyor belt start rolling. Since halting is considered as a routine operation, only the motor need to be turned off. Further, during routine mode, normally the conveyor starts rolling, however, if reset button is not pushed and the
previous mode before exiting the routine mode was halting, the conveyor mode need to start from halting. This condition will avoid an attempt to roll while a product is at the downstream; the condition was represented by the history junction—$H$.

During shutdown mode, the conveyor is expected to receive signal that Robot A has been shut down (represented $convAShutDown$) and no product is detected by the infrared sensors located at both ends of the conveyor. At the same time, no product should be detected downstream of the conveyor, meaning the belt has to be on rolling mode to shut down. However, since the sensors are set at the two extreme ends, product at the middle of the conveyor cannot be detected. Hence, timer delay with value of estimated time for a product to reach downstream of the conveyor is introduced (represented by state $ConvAShutDownDelay$). This would safely end the conveyor process without a product left on the belt.

Based on the stated conditions, four outputs were (re)set: signal for conveyor motor to stop or run, signal to indicate if the conveyor was in either Down, Shutdown and Routine state. Conveyor shutdown status was used as input to Robot B controller for safe shutdown. If for some reason, say stop button is pushed, the conveyor enters Down mode, then the signal $covADwn$ could be used to generate an alarm, thereby stopping the whole manufacturing cell. In return, if an alarm is set by any other machine, in the cell $sysAlarm$ would be set to True, thereby stopping the conveyor (see subsection 3.3.1.4.9 details for system wide alarm).

### 3.3.1.2 Machine Tool Models

In the framework of this thesis, the machine tool models developed were meant to represent the major modes of the machines and monitor their behavior, not to actual run and control the machines when the generated code is downloaded. Using the major modes representation, the status of the machines was predicted and interpreted in a way other models can use.

The three machine tools were modeled in the same way. Lathe machine modes and conditions representation could apply to the two milling machines modes. The lathe machine was considered to have three mutually exclusive modes represented by $LatheDown$, $LatheRoutine$, $LatheShutDown$ states. For safe start up, the lathe model was made to start from $LatheDown$ state, which represented the lathe chuck motor and other components set to stop. Among other conditions, for the lathe to change from the down mode to Routine mode, the following factors were considered to be fulfilled: neither emergency stop, lathe stop, nor manufacturing
cell wide alarm should be True. If either of the aforementioned conditions is not fulfilled, the lathe was expected to enter Down mode.

![Stateflow model representation of conveyor modes](image)

**Figure 19. Stateflow model representation of conveyor modes**

During Routine mode, the Lathe was expected to work between two mutual exclusive modes: represented by states LDoorClosed and LDoorOpen. Even though further decomposition of these states would reveal many child states, but for the purpose of the manufacturing cell control one further step decomposition was sufficient.

Normally, the lathe was assumed to be in closed door state during start up. During this state, the lathe is either machining (represented by the LTurning) or idling mode (represented by the state LIdlling). For safe start up, the lathe was assumed to enter the idle state when door is closed; if an internal signal signifies the turning was done or not started (represented turningDone), the door was expected to open to accept a new stock.

![Stateflow model representation of lathe machine Routine mode](image)

**Figure 20. Stateflow model representation of lathe machine Routine mode**

Further, the assumption was that the lathe door opens for two reasons (thus modes) either to accept a stock or eject a machined product. These modes were represented by states ProdEnteringL and ProdExitingL, respectively; for a product to enter a lathe there should not be
any product on the chuck, and there should be a product on the chuck to enter the product exiting mode.

According to Rule T2, the lathe should remain open until a product enters into the chuck. Similarly the machine should remain open until the product exits. To represent these rules a transition condition \textit{latheLoaded} was used to make a state loop to keep the door open. Further, an infrared signal, notated by LDO\textit{bstacle}, should prevent the closing of the door while the robot is entering or exiting the lathe.

Moreover, the assumption was that the machine tool shuts down only when the door is open, the chuck is not loaded and Robot B is shut down. This means the lathe expects an input signal on the status of the robot. The outputs \textit{doorLopen} and \textit{prodOnL} were used as input signals to Robot B controller, which was expected to use this information to move to and from the lathe.

\textbf{3.3.1.3 Robot A Model}

Robot A model was much easier to model than Robot B in terms of the number of inputs and outputs involved and related connections. Unlike to conveyor and machine tool models, Robot A (and Robot B), assumed to start up with starting mode (represented by state \textit{RAS\textit{t\textit{arting}}}), during which motors are turned on. Once motors are turned on, the robot is should enter the down mode (represented by the state \textit{RA\textit{Down}}); and if neither emergency nor stop button are pushed, and the robot should enter the routine mode (represented by the state \textit{RAR\textit{outine}}). During this mode, the robot is forced to move to the palette to pick up a stock (represented by \textit{MovingToPallet}) and moving to a conveyor to drop (represented by \textit{MovingToConvA}). Whenever there is need to shut down the manufacturing cell, Robot A is shut down first followed by conveyor A.
3.3.1.3.1 Starting-up Mode

Robot A starting-up mode was represented by the state RAStarting. As stated in section 3.2, ABB IRC5 controller uses FBD to invoke actions of a robot, but Stateflow Coder does not support the FBD invocation. Hence, when start main button, represented by RAStartMain, is pressed, the function block IRC_STARMAIN is expected to be triggered; the triggering signal was represented RAStartMainTrig (see section 2.4, Figure 11, and section 4.2.2.3.1 for further details). When start main is done, RAStartMainDone is expected to be automatically set to True. Similarly, when IRC5_START is invoked by RAStartTrig, RAStartDone is expected to be automatically set to true, triggering the motors. When motors start the robot is expected to enter the down mode. The models for the starting up, down and shutdown mode models are the same at that of Robot B. The models are discussed in subsection 3.3.1.4.7 and 3.3.1.4.8, respectively.

Figure 21. Stateflow model representation of major Robot A modes
3.3.1.3.2 Routing
The robot is expected to be at home position during the routine mode startup; otherwise the robot is set to move to home position, represented by the state RAHoming. The robot may receive an alarm or stop signal forcing it to be in down mode; if reset button is pushed, the robot moves back to home position from whichever position it had previously acquired, otherwise, the robot is expected to resume the normal operation. This condition was simplified by the use of the history junction shown in the figure below.

During the first move or in case of reset button, the robot is expected to move to waiting position after home position. Normally, when a pallet arrives with a product(s), the robot should
be forced to move to the pallet to grip from and drop on the upstream end of Conveyor A. But since the conveyor is controlled by its two ends, there is high probability that the dropped product will remain on the target position on the conveyor, due to no pick up of a stock at the downstream of the conveyor by Robot B. In such case, Robot A should be forced to go the waiting position. Also, the robot has to stop to grip or release. Hence, the state \textit{StopWaitAndAct} was used to represent the modes associated with robot stopping, waiting and acting. This mode is further discussed in subsection 3.3.1.4.

Figure 24. \textit{Stateflow model representation of Robot A stop, wait and grip/release modes}

Motion towards the pallet or the conveyors was considered to be more than one step motion. However, for this paper two step motion was considered represent the modes involved. The first motion was represented by \textit{InitialMoveP} and the second one by \textit{StepMoveP}. Similarly the motion to Conveyor A was represented \textit{InitialMoveConvA} and \textit{StepMoveConvA}. For further explanation please see subsection 3.3.1.4.3.

Figure 25. \textit{Stateflow model representation of Robot A motion to pallet and Conveyor A}
3.3.1.4  **Robot B Model**

This model has 27 input, 11 local and 23 output signals, and 49 states. Similar to the approach followed for Robot A, the robot is expected to be on starting mode until all motors are turned on, be on down mode until the robot receives is no system alarm, error, stop, and power relay signals, and be on routine mode until the the robot receives the alarm, error, stop, power signals or system shutdown signal (leading to shut down mode). These modes were represented by states **RBStarting**, **RBDown**, **RBRoutine** and **RBShutDown**.

![Stateflow model representation of major modes of Robot B](image)

**Figure 26. Stateflow model representation of major modes of Robot B**

3.3.1.4.1  **Robot Cycles**

The robot has to transport the stock from Conveyor A to Conveyor B through Lathe, Millings A and B; one cycle of the robot has several modes that occur in sequence. In figure below shows the first cycle of the robot when all machines were not loaded; in this case, the Lathe machine has shorter cycle time than the milling machines. These steps were used for initial modeling of the robot modes. More details were added with time, by considering all possible cycles of the robot.
3.3.1.4.2 Routine Model and Motion Planning

During routine mode, the robot is expected to move to home position for safe start up, represented by state *Homing*, until the target coordinate is reached; upon exit of this mode the robot is given instruction to go to Conveyor A. At the HMI the entry and exit to this mode of the robot is represented by output signal *RBH*. At the moment of exit, the variable *RBMovePriority* was set to 0, to make the next current state *MovingToConvA*. One important note worth mentioning is that the variables *RBmovePriority* and *prodType* were used to guide series of sequence of robot moves; by using these variables the complexity was strictly reduced and execution of another instruction before completing the ongoing task was made impossible. Specifically, variable *prodType* was used to guide the robot to drop a particular product at a particular machine. This was achieved by setting the variables with different numbers only when the system exits a current state.
After the home position, the robot model simulation current state indicator was made to enter the state StopWaitAndAct, from which decision is made to go to either the conveyors or machine tool models. Directly after the robot has reached the home position, a new position, waiting position was set; motion to this position was represented by state RBWaitingForInfo and status indicator output signal by RBWFI, as shown in below.
3.3.1.4.3 Motion-to-Conveyors Model
According to Rule R5, the robot moves to Conveyor A, if the lathe door is open, there is no product on the lathe and the robot, and there is product to grip on the conveyor. If this condition is not fulfilled the robot is forced to wait at its waiting position. But if the condition is fulfilled, then the robot picks up the material from the conveyor by following two step motion, which were represented by InitialMovingConvA and StepMoveConvA. The first state represented the motion from waiting position to the conveyor, and the second state the short distance motion to a position very close the product position. Similarly, the robot moves to Conveyor B, if there is product on the robot and no product at the target position on the conveyor upstream. Then it follows the two step motion as in motion to Conveyor A.

![Stateflow model representation of motion of Robot B to conveyors A and B](image)

3.3.1.4.4 Wait-Stop-and-Act Model
When the two step motion is completed, to force the robot to stop (either to grip or release), the state RBStop was used. When no end effector coordinates are set, the robot is expected to stay above the target position, for instance, at the downstream of Conveyor A; at this point, the robot decides whether to grip or release. The states RBRelease and RGBrip with no coordinates were introduced to represent the action. If there is no product loaded on the robot, the gripeing motor of the end effector is turned on (this was done by setting RBEndEffectorMotor to True). When the end effector motor is actuated, the robot expects a signal from an internal sensor that a product is gripped (notated by condition prodOnRB).

This expected signal was used to dictate an action to release the stock on the lathe. The move priority number was set to 2, signaling the next motion is to lathe to release the product, neglecting the condition movePriority==8 just below the RGBrip state.
3.3.1.4.5 Motion-to-Lathe Model

The robot moves to the lathe for two reasons either to grip or release; the two moves were represented by the states $LMoveToGripp$ and $LMoveToRelz$, and the entry/exit signals to/from the two modes corresponding to the two states were named $LMTG$ and $LMTR$. While gripping from Conveyor A, the move priority had been preset to 2, the state $LMoveToRelz$ became active. During this mode, two set of coordinates were provided as targets. During the exit of the state $LMoveToRelz$, move priority would be set to 9. To exit this state/ corresponding mode, the robot must have reached the target. To dictate the robot to stop and stay beside the chuck, the state $StopWaitAndAct$ with no target coordinates was used. Because the turned product would be on the robot this time around, the robot is expected to enter the release mode, during which the end effector motor is turned off, dropping the product.
The robot has to check whether Lathe, Milling A or Milling B are open and loaded; according to Rule R6, the robot is set to move to Milling A (or Milling B, with priority given to Milling A) to set a free position on the milling chuck for the turned product on the Lathe. But if there is no product on the lathe, then robot should be set to move to Conveyor A, to feed the Lathe from the conveyor; if none of the conditions are satisfied, then the robot is set to move to waiting position (with movePriority set to 10). These requirements were represented by conditions just below the state RBRelease.

For a case when the first product of the first batch is on Lathe, either of the milling machines are open (or both are open) and not loaded, the robot should return back to the lathe to grip the turned product. Also, a mandatory movement towards the lathe with purpose to grip should be enforced when the robot drops the product on Conveyor B from the either of the milling machines; according to Rule R6, the robot has to feed the empty milling machine from the lathe. But if the lathe door is closed the robot has to go to waiting position; this was done by setting the movePriority to 10 first and to 1 during the state WaitingForInfo.

3.3.1.4.6 Motion-to-Milling Machines Model
The robot should move to the lathe to grip, if either of the succeeding machines is ready, in accordance to Rule R5. This was done by setting movePriority 8 at state LMoveToGrip. This priority number led to another condition that checks the state of the doors of milling machines (see the conditions just below state RBGrip in Figure 31). With priority given to Milling A, the robot should move the turned product from the lathe to either of the millings. The motion
during this mode was represented by state \textit{InitialMoveMARlz} (or \textit{InitialMoveMBRlz}) followed by the short distance move mode, represented by \textit{StepMoveMARlz} (or \textit{StepMoveMBRlz}).

![Figure 33. Stateflow model representation of motion of Robot B to Milling A](image)

At the end of this mode, the move priority was set to 0, instructing the next move towards the Conveyor A to feed a stock to the lathe. In case of a similar cycle time of the machines, either of the millings is expected to be ready ahead of the lathe. Since entering the lathe to release would set the priority to 9, the set of conditions just below the state \textit{RBRelease} would be used to choose either of the millings.

![Figure 34. Stateflow model representation of motion of Robot B to Milling B](image)

Once the decision is taken to which milling to move, the robot should follow the two step move to grip, as explained in subsections above. After completing the two step motion, the robot
should move to Conveyor B; if there is no product on the target position at the conveyor upstream, the robot releases at the target, sees section 3.3.1.4.4. To instruct this move \textit{RBMovePriority} was set to 7.

### 3.3.1.4.7 Down Model

According to \textit{Rule G2}, during \textit{Routine} mode, if any signal associated stop, emergency stop, system alarm, error, ranch chain open or power cut received the robot should be forced to stop. Specifically if the robot receives signal associated with emergency, ranch open or error, there is a need to reset the robot before resuming the work (ABB, 2010). Hence, the state \textit{RBStoping} was used to trigger the stopping action, which would later be used for invocation of the function block \textit{IRC5\_STOP}. To reset the emergency stop, error, and ranch signals, \textit{RBRstEmStpTrig}, \textit{RBRstErrTrig}, and \textit{RBRstSftyTrig} were used, respectively, to trigger their corresponding function blocks. To indicate that the robot is ok, the state \textit{RBResetDone} was used.

![Stateflow model representation of Robot B down mode](image)

**Figure 35. Stateflow model representation of Robot B down mode**

### 3.3.1.4.8 Shutdown Model

The manufacturing cell should be shut down in orderly manner and when there is no work in progress, in accordance to \textit{Rule G5} and \textit{G5.1}. As shown in Figure 26, these requirements, were represented by introducing the condition \textit{sysShut}, \textit{!prodOnCADwnStrm} \&\& \textit{!prodOnL}, \textit{!prodOnMA}, \textit{!prodOnMB}, \textit{!prodOnCBUpStrm} \&\& \textit{!prodOnRB}.
According to Rule R8, the robot has to move to home position before shutting down. To represent these requirements, the state RBMoveToHome was used to provide the target position. Further, the robot motors shut down instruction was given by RBMotorOffTrig; when the motors are turned off the RBMotorOffDone would be set true. See Appendix III.

3.3.1.4.9 Integrating Representative Models

Checking that all models individually simulated, given the inputs and outputs of the individual machine models, there is absolute need to check further if the models of the machines can work synchronously, with manufacturing cell perspective. The assumption during integration is that there is a communication PLCbus among the machines; the assumed communication lines were indicated by the connection lines between Stateflow charts. For instance, signals from Conveyor A upstream and downstream infrared sensors connected to Robot A and Robot B respectively.

While bottom-up-modular development approaches is acceptable method, as far as there is a need to know the status of other machines, in terms of alarm, error or emergency stop, communication is needed to/from every machine. Three options were considered to handle this task. The first option was to connect every machine to every other machine; the second option was to use an additional orchestrator controller; and the last option was to develop a model in such way all communication is done with Robot B.
In this paper the second option was chosen for two reasons. First option needs several communication lines and was labeled as inefficient. And the third option would make the model not truly modularized. So another orchestrator controller was used. The logic that can be downloaded to this controller was developed in truth table format.

In the truth table (represented by `alarmTable`), if any of the machines emergency stop, stop, or any other button is pressed, system wide alarm is triggered. For instance, when Conveyor A, for some reason, enters `Down` mode, then truth table condition `isConvDwn==true` would be satisfied (see Figure 39), hence the corresponding decision (D2) value was set to `True`. If none of the machine is down mode then, action `A1` would become `True`.
3.3.2 Top-Down Model Development Approach

The need for top-down model development approach was discussed in subsection 2.3.5.2; the argument is that the two robots interact with five other machines and the five other machines depend on each other at least by system wide alarm signal. Hence, the paper considered the application of this method from the manufacturing system holistic perspective. This model has 77 input, 50 local and 75 output signals. What is discussed in the subsections below is deliberately shortened for sake of brevity.

The approach follows the same rules and requirements used during bottom-up-modular model development approach. To develop the model using this holistic approach, the manufacturing cell was assumed to exist in either of the three modes: down mode when not working, routine mode when working normally, and shutdown mode when shut down. The assumed modes were represented by the states SystemDown, SystemUp, and ShutDown, respectively.
3.3.2.1 Truth Tables

Since every machine affects every other machine, four truth tables were used to represent part of the system logic. The four truth tables have similar logic structure but use different input and output signals. The truth tables alarmTable, stopTable, and powerRelayTable were configured in such a way to accept direct input signals from Manual Switch blocks (which represent HMI buttons or sensors). But shutTable was configured to receive output signals from the substates. For example, from child state ConvBShutDown the value of the output signal isConvBShut was used as an input to the shutTable.
3.3.2.2  Manufacturing Cell Routine Model

All machines should work in parallel, with an exception during the startup and shut down. But in terms of running the program, the machines logic should work in parallel. If and when the machines receive appropriate signal, the machines respond. To represent these requirements, seven models were put in parallel (AND-decomposition). Using this representation the seven models were executes and became sibling current states at the same time.

Figure 42. Manufacturing cell System Routine model

Under the parent states shown above, similar to the states used during Bottom-up-modular model development approach, the routine and shut down states were put under their
corresponding parent states. For instance, the states RARoutine and RAShutDown were incorporated in the parent state RobotA. If and when the manufacturing cell is stopped and there is a need to resume the operations, the machines should start from the modes previously existed. To represent this each parent state was endowed with the history junction.

![Figure 43. Stateflow model representation of Robot A routine and shut down modes](image)

### 3.3.2.3 Manufacturing Cell Down Model

The manufacturing cell works with no buffer, only two robots to/from transport to five different machines; so if any of the machines fail or stop, all the machines should stop, in accordance to *Rule G2*. To represent models were used to represent the seven machines’ down modes. Similar to the child states of the state RBDown, discussed in subsection 3.3.1.3.7, were incorporated in both robots. The conveyors’ motors were set to be turned off and machine tool’s status indicators were set to represent their corresponding Down modes.

![Figure 44. Stateflow model representation of manufacturing cell down mode](image)

### 3.3 Summary

The chapter presented most of the models used to represent the manufacturing cell modes. First rules and requirements were presented for each machine set. Based on the rules and requirements, nine models were developed using two model development approaches: Bottom-up-modular and Top-down model development approaches. Since all machines operate semi-
autonomously, model for each and every machine was developed independently which were later integrated to represent the manufacturing cell modes. However, from perspective of Robot B, the manufacturing cell used was coupled due to incidence of alarm considerations and close interaction of the robots with the other machines, the top-down model development approach was used. Using this approach all major modes of the manufacturing cell were considered first and then more details were added by decomposing the modes. Every model presented in this chapter was checked for correctness and simulated.
Chapter Four

Virtual Verification and Validation

This chapter presents verification and validation of the nine models presented in the last chapter and codes generated from the models. The chapter is divided into two parts. The first part presents and discusses model and code verification results; and the second part presents and discusses the model and code validation results.

4.1 Model and Code Verification

Model and code verification was done in accordance of the arguments presented in subsections 2.5.1 and 2.8.2. The following subsections discuss and summarize the verification processes.

4.1.1 Model Verification

4.1.1.1 Model Advisor Output Report

When simulation was run the Stateflow Parser and Stateflow Debugger automatically checked for syntax errors, transition conflicts, state inconsistencies, data range violation, and cyclic behavior. Once the Stateflow Parser and Stateflow Debugger reported that there were no errors, the model advisor was utilized.

When the Model Advisor was run, all models passed 83 different tests for by product run and 101 different tests by task run, all with 0 numbers of fails. Moreover, PLC code compatibility for PLC Code generation was checked for all charts before codes were generated; the report “PLC compatibility check passed” was received for all models.

4.1.1.2 Design Verifier Analysis Report

Several independent tests were run using Design Verifier tools: Check Subsystem Compatibility, Detect design errors for subsystem and Running Prove properties for subsystem tools reported “Chart is compatible with Simulink Design Verifier”, “Design error detection completed normally”, “Property proving completed normally” respectively. The Run test for subsystem option reported result of several test cases. The main results are shown in Table 3 and a sample test case in shown in Figure 45.
Table 3. Simulink Design Verifier Tests Report

<table>
<thead>
<tr>
<th>Machine Model / Design verification part</th>
<th>Satisfied Objectives</th>
<th>Proven Unsatisfiable</th>
<th>Total test Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conveyor Models</td>
<td>52</td>
<td>4</td>
<td>56</td>
</tr>
<tr>
<td>Machine Tool Models</td>
<td>80</td>
<td>1</td>
<td>81</td>
</tr>
<tr>
<td>Robot A Model</td>
<td>191</td>
<td>2</td>
<td>193</td>
</tr>
<tr>
<td>Robot B Model</td>
<td>423</td>
<td>3</td>
<td>426</td>
</tr>
<tr>
<td>System Alarm Truth Table</td>
<td>22</td>
<td>8</td>
<td>30</td>
</tr>
</tbody>
</table>

Running test for subsystem, however, several tests objectives were found **Unsatisfiable**; for instance, when conveyor shutdown delay is introduced using the state `ConvAShutDownDelay`, the condition is `convADelayDone` was unreachable objectives (“Objective Proven Unsatisfiable”). Similar cases for Robot A and Robot B were reported. The conditions that were not reachable in Stateflow model were later assigned corresponding values in CoDeSys environment. For the case of conveyor codes, Function Blocks like `Timer On (TON)` can be invoked by putting the lines below at the end of the program.

```plaintext
timerInst: TON;             (*timerInst Declaration *)
timerInst(in:=timerTrig, pt:=t#5s);  (* timer trigger*)
convADelayDone:=timerInst.Q;   (*sets output of timer*)
```

Invocation of Function Blocks related to robot, like start (`IRC_START`), is discussed subsection 4.2.2.3.2 and the proposed declaration and lines for invocation of the blocks are presented in Appendix III.

In case of truth tables, the last columns of the truth table like in Figure 41 were not applicable for different tests, hence **Unsatisfiable** objectives. Furthermore, since no fault analysis was done, possible abnormal behaviors of the machines were not considered. For instance, when Robot B moves to the Lathe to release a product, the robot is expected to successfully release the product; so the model condition if and when a product isn’t released successfully was not considered. Hence, in model verification test, the condition that related to unsuccessful attempt to release a product became **Unsatisfiable** objective, increasing the total number of **Unsatisfiable** Objectives.
In case of Robot A and B, model verification with several test cases was important for finding signals leading to deadlock. Since all possible conflicting signals, dead logic were checked, the probability of deadlock was reduced.

4.1.2 Code Verification

Code verification was done using Simulink PLC Coder by utilizing the “Generate, import and verify code for subsystem” option. The Coder automatically declares variables, generates code and test cases. The number of lines, POUs, and global and local variables of the generated codes are reported as in Table 4.

**Table 4. Statistics of generated PLC codes from 5 different models**

<table>
<thead>
<tr>
<th>Model Type</th>
<th>Lines of PLC code (without comments)</th>
<th>POUs</th>
<th>Global Variables</th>
<th>Local Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conveyor code</td>
<td>111</td>
<td>2</td>
<td>11</td>
<td>30</td>
</tr>
<tr>
<td>Machine tool code</td>
<td>240</td>
<td>2</td>
<td>11</td>
<td>32</td>
</tr>
<tr>
<td>Robot A code</td>
<td>412</td>
<td>2</td>
<td>11</td>
<td>69</td>
</tr>
<tr>
<td>Robot B code</td>
<td>919</td>
<td>2</td>
<td>11</td>
<td>116</td>
</tr>
<tr>
<td>System Alarm Code</td>
<td>17</td>
<td>3</td>
<td>11</td>
<td>17</td>
</tr>
<tr>
<td>Monolithic Code</td>
<td>3054</td>
<td>6</td>
<td>11</td>
<td>335</td>
</tr>
</tbody>
</table>
The test cases for the code are generated based on Stateflow input signals collected during simulation. Hence, proper choice of Simulink blocks that were connected to the Stateflow block was important for verification. For instance, the test setting in Figure 12 and 47, the test cases were all True or all False for given input signals, since a constant value is connected to the Stateflow. To collect a reasonably variant of input signals for the test bench, a block with changing value, like the Pulse generator block with different pulse widths and periods, was used. These settings generated different set of input signals for model simulation.

Furthermore, Simulink PLC Coder automatically recorded the set of input and output signals; the input signals were used as input signals for the generated code by the test bench and the output of the generated code comparison with the input and output signals from CoDeSys. Whenever there is need to change the number of input signal within sets, the simulation time was incremented and decremented. Using this approach, the generated codes from the nine models were verified.

![Part of test bench input values of lathe code in CoDeSys Environment](image)

### 4.2 Model and Code Validation

As stated in section 2.5 and 2.8, the validation is done to check whether the needs of the user can be satisfied. It’s also stated that rules and will be used as requirements. In line with these arguments, models were checked against the developed rules. The table below shows which rules should be used to validate a model/code. For example, for conveyor A, general Rules G1 to G5 and conveyor specific Rule C1 only apply. See Appendix IV for examples functionality test cases that were used for code and model validation.
4.2.1 Model Validation

Each model was validated by visualization method by changing the input variables and studying the current state. When the current state is a child state, it also in parent state; hence, list of all terminal states of a Stateflow model can indirectly represent the parent states. To make each terminal state a current state, one or more input variables were (re)set to True or False. Manual switch blocks were used to change input values.

![Figure 47. Simulink Manual Switch and other blocks connected to a Stateflow chart](image)

Using the above approach, each and every model was validated in accordance to functionality test table shown in Appendix IV. The screen captures most of the models of the seven machines were presented in Chapter three. However, since modularized validation does not guarantee overall modal validity (Balci, 2010), top-down model development approach was used to increase the degree of confidence on the validity of the models.

### Table 5. Model/Code and Rule relationship table

<table>
<thead>
<tr>
<th>Rule</th>
<th>G1</th>
<th>G2</th>
<th>G3</th>
<th>G4</th>
<th>G5</th>
<th>C1</th>
<th>T1</th>
<th>T2</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
<th>R7</th>
<th>R8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conveyor Model/Code</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Machine Tool Model/Code</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Robot A Model/Code</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Robot B Model/Code</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Monolithic Model/Code</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
4.2.2 Code Validation

Validation, as aforementioned, is satisfying the needs of the customer, in this case, the needs of the process engineer. The code validation was done by providing set of desired input signals that produce specific expected output(s) and by random set of input signals that produce any output signal. In the latter case validation is done by studying the output(s) in terms of the random set of input signals.

Further, to study the outputs of the signals produced from code, the visualization method presented in subsection 2.8.3.2 was used. Since Simulink PLC Coder sets the generated PLC code in the form of Function Block and/or simple function, the input variables were (re)set using the Integrated Visualization Editor. A SFC with ST was used to continuously invoke the code. In all validation steps, SFC, similar to the one in figure below, was used to separate the Step Init and Step Routine.

![Sequential flow chart for lathe code validation: during Routine Step](image)

Normally, when the codes were first run, initial part of the code was executed (SS_INITIALIZE part); for instance in a setting like Figure 48 and 49, Lathe Function Block was invocated by latheInst. When seqStart was set to True on the Visualization Editor (whose commands were already connected to with variables), the Step Routine became active. During Step Routine, the parameter SS_STEP, among other inputs parameters were used. The parameter SS_INITIALIZE
was used to indicate that the program (re)initiating. During program initialization all parameters were *False*, unless deliberately set to *True*.

After several tests, and notating and re-notating input parameter names, the easiest way for function invocation was to copy Simulink PLC Coder’s input parameters that were used for test bench. These names start with a “cycle_” followed by the name of input variables used during modeling.

![Simulink PLC Coder's input parameters](image)

4.2.2.1 **Conveyors’ Code Validation**

The following figure shows result of functionality test case 8 shown in Appendix IV; the case imitates a scenario when start button was pressed and a product was sensed by an infrared sensor at the upstream of the conveyor. The output was that a signal was sent to turn on the motor and indicating the status of the conveyor on *Visualization Editor*. The output was labeled as valid for satisfying the *Rule C1*. The following lines of code were introduced in at the end of the programs lines to set a timer delay during shutdown.

```plaintext
  timerInst: TON;             (*timerInst Declaration *)
  timerInst(in:=timerTrig, pt:=t#10s);  (* timer trigger*)
  convADelayDone:=timerInst.Q;   (*sets output of timer*)
```

![Functionality test of the conveyor code](image)
4.2.2.2 **Machine Tools’ Code Validation**

The following Figure show functionality test cases 11. In this test the lathe input buttons turning done, chuck loaded and door limit switch (open) were set to *True*, resulting color change associated with product on lathe and the safety door open. Further example, when buttons associated with stop, power relay or alarm were set, the gray color of “Lathe Down” changed to green (not shown in the figure below).

![Lathe Code Validation Diagram](Image)

**Figure 51. Functionality test of the Lathe code**

4.2.2.3 **Robots’ code validation**

4.2.2.3.1 **Robot A’s code validation**

The figure below shows a typical functionality test case. The green colors indicate that the robot is on routine mode, during which product on a pallet is picked and moving towards Conveyor A. When button “RA on Target” was bushed the color indicating robot gripping turned off, indicating that the product dropped at the Conveyor and color indicating robot moving to pallet turned on, (not shown here). Several other tests have been performed to validate the code.

![Robot A Code Validation Diagram](Image)

**Figure 52. Functionality test case of the Robot A code**
4.2.2.3.2 Robot B’s code validation

To validate the code several outputs were studied for given values of input. In the case shown in the figure below, the robot is moving to Milling A while holding the product, which is a normal routine operation (indicated by green). These outputs were generated because the variable product on RB, Milling A open were set True, product on Milling A set to False, product Type to 1 and priority number to 4. Several tests were done in accordance the functionality test cases.

![Robot B Code Validation Diagram](image)

**Figure 53. Functionality test case of the Robot B**

Furthermore, in line with the argument of sections 2.4 and 3.2, putting the following lines at the end of the generated PLC code, the robot program is expected to work without any errors. Since the scope of the paper is limited to virtual validation, the following lines were not downloaded to a controller. These lines are part of code shown in Appendix II.

```
(*Declaration part*)
RBStartInst : IRC_START;
RBStartMainInst : IRC_STARTMAIN;
RBMotorsOnInst : IRC5_MOTORSON;

(* The first few lines that should be put at the end of the program*)
RBStartMainInst(RBStartMainTrig, cont);  (*Triggers start Main FBD *)
RBStartMainDone:=RBStartMainInst.DONE;  (*Set the output of instance of FBD*)
RBStartInst(RBStartTrig);               (*Triggers start FBD *)
RBStarDone:=RBStartInstDONE;
RBMotorsOnInst(RBMotorsOnTrig);         (*Triggers Motors on FBD *)
RBMotorsOnDone:= RBMotorsOnInst.DONE;
```
4.2.2.4 Validation of Monolithic Code

As discussed in the previous sections a modularized mode validation cannot assure the validation of whole model. But validation of all machine modules at the same time would bring close to a complete validation. In traditional code development and validation environment, one of the weaknesses of monolithic code is associated with complexity for debugging and maintenance. But using model-based approach, more than 3000 lines of code were easily validated. When the generated result gave unexpected results, errors were not checked in the code, but at the monolithic model, which was much easier to trace.

When validating the monolithic code, the several inputs signals, like sysAlarm and isRBShut that were used in a modular validation repeatedly in each and every machine, were used only once. Basically, only the buttons associated with a machine were used, and only once. The figure above was set by the buttons shown in the figure below. This result represents when Robot B is moving to grip the a product from Conveyor A, lathe door open, Robot A moving to pallet to grip a stock, Milling Machine A open while waiting for a product pick up and Milling Machine B open while waiting for product.
4.3 Summary

This chapter presented the methods used for verification and validation of the developed models and generated codes. For model verification several MATLAB tools like Stateflow Parser, Model Advisor, and Design Verifier were used; the results show that the developed models were compatible for PLC code generation and meet modeling standards. States and conditions within the models that were not applicable for test cases were pointed out. Once the models were verified, validation was done using Manual Switch blocks for changing the values of the input signals, and flow diagram simulation and Scope block for output visualization. The verified and validated models were then used for code and test bench generation. To study the generated code, Sequential Flow Chart and Visualization Editor were used. In the Visualization Editor several buttons (that represent input signals) and visualization objects (that represent the output signals) were used. Reference input and output signals were used for model and code validation.
Chapter Five

Conclusion and Recommendation

5.1 Conclusion

There is a great need for automating, increasing flexibility and configurability of manufacturing lines. Several researchers presented modeling, code generation, verification and validation techniques, separately and with different software. To the best knowledge of the author of this thesis, however, none of the researchers presented PLC programming of a highly coupled manufacturing cell by utilizing the Simulink PLC Coder. Moreover, a holistic steps starting from modeling to PLC code validation using Simulink PLC Coder (and CoDeSys) has not been studied well.

In an attempt to contribute to the holistic approach, this paper considered two robots, two conveyors, a lathe and two milling machines as the main elements of a manufacturing cell. To represent the modes of the manufacturing cell, nine models were developed, verified and validated—7 models for each machine and a model for system wide alarm using Bottom-up-modular and Top-down model development approaches. From these models, total of more than 5000 lines of PLC code were (re)generated from the models, which were later verified and validated.

The main contribution of this paper is to clearly show how to virtually model, generate PLC code, verify and validate for purpose of controlling a highly automated manufacturing system, thereby prove applicability of the methods for the stated objective. The conclusion and contribution of this thesis are summarized as follows.

a. Bottom-up-modular and Top-down model development approaches can be used to model all manufacturing system modes. To achieve each machine was modeled independently, which each model was later integrated with other machine models. In top down approach, all major modes of the manufacturing cell were first considered and later decomposed to add details. The two approaches gave the similar result.

b. Models and generated PLC codes for control systems can be virtually developed, verified and validated. Models were developed using two methods stated in the above; models
were verified using Simulink Model Advisor and Design Verifier; generated codes were verified by comparing Simulink simulation and generated code outputs for a given set of inputs (this was done by Simulink PLC Coder); models were validated by studying the flowchart simulation in Stateflow environment and other Simulink blocks; and PLC codes were validated by studying the outputs using 2D visualization in CoDeSys environment. Validation was done by referring the developed set of rules and requirements.

c. A generated PLC code, in structured text, can easily be retouched to instantiate Function Block(s). Simulink PLC Coder generates codes in structured text format; however, many functions of control system are instantiated by calling a Function Blocks. To circumvent this limitation, several generated codes were studied and all generated codes were found to have exactly one pattern. The value of the actual output is set at the end of the generated programs. Hence, considering the functions blocks at states during modeling and adding two lines of code per Function Block gives the desired function.

d. There exists limited literature on manufacturing system modeling using the Simulink PLC Coder, verification and validation. The paper presented a step-by-step virtual development, verification and validation of models and generated PLC codes. The approach illustrated the benefits of following the steps. The paper has also shown that monolithic PLC codes can easily debugged, which would have been difficult using conventional coding approaches.

5.2 Recommendation

Most of the stated delimitation can be extended for future work. This paper considered limited numbers of sensors; the developed models can be enriched by considering multiple sensors and make more practical decision based on the status of the sensors. With appropriate number of sensors, the developed models can be extended to scenarios as shown in the robot models (which operations take time to be completed). Moreover, how timers can be modeled and generated code retouched was presented; this can be extended for scheduling manufacturing cell operations using a generated code.

Now that the manufacturing cell models and codes are developed, verified and validated the generated codes can also be downloaded to real controllers. Further, 3D emulation and real time testing for better visualization of the operational behaviors can be done using the
developed models. Moreover, data communication between PLC program and robot specific program was highly neglected for robot programming. This gap can be filled by presenting PLC data capturing techniques, like robot coordinates.

To sum up, today's manufacturing organizations have the option to automate, increase flexibility and configurability of manufacturing lines in order to cope with several constraints. Success on this option is highly dependent on how quickly a PLC code is developed, verified, validated. To meet the needs, a virtual modeling, code generation, verification and validation technique can significantly reduce time and associated costs. Given the benefits, the research was conducted to present a way of meeting the needs.
Appendices

Appendix I [A Selected Part of Model developed by following Top-down Model Development Approach]
Appendix II [A Selected Code: Conveyor Code]
IF NOT prodOnCADevsStr THEN
  is_CONVRoutine = Conveyor_IH_CONV_Routine;
  was_CONVRoutine = Conveyor_IH_CONV_Routine;
  b_CONVMotor = TRUE;
END_IF;
ELSEIF prodOnCADevsStr THEN
  is_CONVRoutine = Conveyor_IH_CONV_Motor;
  was_CONVRoutine = Conveyor_IH_CONV_Motor;
  b_CONVMotor = FALSE;
END_IF;
END_CASE;

IF convADelayDown THEN
  is_CONV = Conveyor_IH_CONV_ShutDown;
  b_CONVMotor = FALSE;
  b_CONVShut = TRUE;
ELSE
  b_CONVTrig = TRUE;
END_IF;
END_CASE;
convA = b_CONVMotor;
convA_Routine = b_CONV_Routine;
convADev = b_CONV_Dev;
is_CONVShut = b_CONVShut;
isProdOnCADevsStr = b_isProdOnCADevsStr;
isProdOnCADevsStr = b_isProdOnCADevsStr;
timesTrig = b_timesTrig;
END_CASE.
Appendix III [Robot B’s Adjunct Code for Function Block Invocation]

(*Declaration part*)

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBStartInst IRC_START</td>
<td></td>
</tr>
<tr>
<td>RBSysResetInst IRC5_SYSRESET</td>
<td></td>
</tr>
<tr>
<td>RBStartMainInst IRC_STARTMAIN</td>
<td></td>
</tr>
<tr>
<td>RBMotorsOnInst IRC5_MOTORSON</td>
<td></td>
</tr>
<tr>
<td>RBMotorOffInst IRC5_MOTORSOFF</td>
<td></td>
</tr>
<tr>
<td>RBStopInst IRC5_STOP</td>
<td></td>
</tr>
<tr>
<td>RBEmStpInst IRC5_RESETESTOP</td>
<td></td>
</tr>
<tr>
<td>RBRstErrInst IRC5_RESETERR</td>
<td></td>
</tr>
<tr>
<td>RBSftyInst IRC5_RESETSMSTOP</td>
<td></td>
</tr>
<tr>
<td>RBStatusInst IRC5_STATUS</td>
<td></td>
</tr>
</tbody>
</table>

(*At the end of the program the following lines should be put*)

RBStartInst(RBStarTrig);    RBStarDone:=RBStartInstDONE;
RBSysResetInst(RBSysResetTrig);   RBSysResetDone := RBSysResetInst.DONE;
RBStartMainInst(RBStartMainTrig,cont);      RBStartMainDone:=RBStartMainInst.DONE;
RBMotorsOnInst(RBMotorsOnTrig);   RBMotorsOnDone:= RBMotorsOnInst.DONE;
RBMotorOffInst(RBMotorOffTrig);   RBMotorOffDone:= RBMotorOffInst.DONE
RBStopInst(RBStopTrig);    RBStopDone := RBStopInst.DONE;
RBEmStpInst(RBEmStpRstTrig);   RBEmStpDone := RBEmStpInst.DONE;
RBRstSftyInst(RBSftyTrig);    RBRstSftyDone:=RBSftyInst.DONE;
RBStartInst(RBStarTrig);    RBStarDone:=RBStartInst.DONE;
RBStatusInst (enableRBStatus);   RBRunchOk:= RBStatusInst.RUNCHOK;

RBErro:= ((RBStatusInst. PWRFAILERR) OR (RBStatusInst. RGNDISTERR ) OR (RBStatusInst. EXECERR) OR (RBStatusInst. PRODERR) OR (RBStatusInst. MOTSUPERR) or RBStatusInst.RUNCHOK)
### Appendix IV [Selected Functionality Test Cases Table with Input Signal, Output Signal and Decision*]

<table>
<thead>
<tr>
<th>Functionality Test case #</th>
<th>Rule #</th>
<th>Input Signal</th>
<th>Output</th>
<th>Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>G1</td>
<td>All inputs fall</td>
<td>All outputs fall</td>
<td>Valid</td>
</tr>
<tr>
<td>2</td>
<td>G1.1</td>
<td>seqStart</td>
<td>lathDown; latheDown; latheDown; latheDown</td>
<td>Valid</td>
</tr>
<tr>
<td>3</td>
<td>G2</td>
<td>ConvAStart + convAEmStop &amp;&amp; sysAlarm &amp;&amp; lathPowerRelay</td>
<td>RARoutine + RAStop &amp;&amp; RAError &amp;&amp; RAStop</td>
<td>convADown; latheDown; latheDown</td>
</tr>
<tr>
<td>4</td>
<td>G3</td>
<td>ConvAStart + convAStop + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARoutine + convARouting...</td>
<td>Valid</td>
<td></td>
</tr>
</tbody>
</table>

*“*+*” means “followed by”. Actual code validation reference names are deliberately neglected for brevity.
References

1. 3S. (2007a). Visualization Supplement to the User Manual for PLC Programming with CoDeSys 2.3, Smart Software Solutions GmbH,


